Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

AQUILA. Assistant for Quantum and Unified Intelligent Layered Architectures

Francisco Javier Espada

franciscoj.espada@uclm.es

University of Castilla-La Mancha

Fernando Martín-Martín

University of Castilla-La Mancha

Ana Díaz-Muñoz

University of Castilla-La Mancha

Ricardo Pérez-Castillo

University of Castilla-La Mancha

Research Article

Keywords:

Posted Date: November 24th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11452937/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

AQUILA. Assistant for Quantum and Unified Intelligent Layered Architectures

Francisco Javier Espada[1*], Fernando Martín-Martín[1*], Ana Díaz-Muñoz[1*] and Ricardo Pérez-Castillo[1*]

[1]Faculty of Social Sciences & IT, University of Castilla-La Mancha, Talavera de la Reina, Spain.

*Corresponding author(s). E-mail(s): franciscoj.espada@uclm.es; fernando.mmartin@uclm.es; ana.diaz@uclm.es; ricardo.pdelcastillo@uclm.es;

Abstract

Achieving quantum utility today relies on the effective design and optimization of hybrid quantum-classical software architectures. Beyond hardware constraints, a major challenge of these architectures is the absence of quantum software engineering methodologies, including best practices and design patterns, which leads to suboptimal development practices. The use of LLMs has demonstrated remarkable success in conventional software engineering tasks, but they often encounter significant limitations in constantly evolving domains such as Quantum Computing. To bridge this gap, this paper introduces AQUILA, a framework based on Retrieval Augmented Generation to elevate the quality, efficiency and design of hybrid software architectures. The system combines a dynamic knowledge base with a prompt template system that ensures the generation of highly precise, context-aware architectural solutions. Operationally, the proposed system leverages LangChain for orchestration, Ollama for secure and local LLM execution and external APIs for high-capacity inference. AQUILA has been validated across two phases: retrieval performance via RAGAs metrics and a generation quality evaluation, contrasting an LLM-as-a-judge approach with human expert consensus. The results indicate effective context retrieval (achieving an F1-score of 0.822), while showing that AQUILA consistently improves response quality over the LLM without context. These findings highlight the potential of domain-grounded retrieval to support quantum software engineers and provide a practical foundation for adopting reliable hybrid architectures in industrial contexts.

Keywords: Hybrid Software Architecture, Quantum Software Engineering, Retrieval Augmented Generation, LLM

1 Introduction

Quantum Computing (QC) has evolved from theoretical explorations toward an era known as Noisy Intermediate-Scale Quantum (NISQ), a term coined by Preskill [1]. In this landscape, the development of hybrid quantum-classical systems has emerged as the most viable strategy for achieving quantum utility today [2]. Under the premise that "*Quantum is Coming*", leading companies such as IBM emphasize that the success of this technology will depend not only on hardware advancements but also on the ability to integrate these capabilities into existing workflows.

Algorithms such as the Variational Quantum Eigensolver (VQE) and the Quantum Approximate Optimization Algorithm (QAOA) require tight integration between classical computing and quantum software. However, designing and optimizing these architectures represents a major challenge due to the rapid and constant evolution of hardware specifications, error mitigation strategies, and design patterns.

The design, development, and optimization of hybrid software architectures constitute a first-class architectural challenge. The current state of the field faces a significant hurdle where the primary difficulty lies in engineering decision-making. Designing hybrid software requires identifying which components should be delegated to quantum backends and which should remain in the classical domain. This necessitates defining scalable architectures that ensure maintainability and minimize the impact of integrating new quantum functionalities.

In addition to these challenges, there is a critical need to apply clear methodological frameworks, design patterns, and best practices for creating hybrid software systems. These software engineering decisions, compounded by hardware constraints, make the development of efficient and robust software a highly complex task.

In response to the complexity of hybrid software design, the use of Large Language Models (LLMs) has begun to be explored as a promising solution across various areas of quantum software engineering, such as automatic code generation [3], code analysis [4, 5], and quantum algorithm repair [6], among others. LLMs possess remarkable capabilities in software engineering [7], but they often fall short in specialized domains like QC. Furthermore, the reliance of these models on vast amounts of training data frequently leads to hallucinations [8], which can result in architectural advice that may seem plausible but lacks real-world alignment.

To address these challenges, this paper presents AQUILA (*Assistant for Quantum and Unified Intelligent Layered Architectures*), a Retrieval-Augmented Generation (RAG) system and support tool designed to generate hybrid (classical-quantum) software architecture designs and their underlying code. The objective is to propose or enhance high-level designs for these hybrid architectures regarding various quality attributes. Unlike standard LLMs, which may suffer from hallucinations or lack updated technical depth, AQUILA utilizes its own database as context. This knowledge base can be updated with the latest definitions of best practices, design patterns, and examples of hybrid architectures and code. This allows the system to obtain the enriched context necessary to generate accurate responses better suited to quantum software engineering practices. In this regard, AQUILA is designed to address fundamental questions such as structural flaw detection, integration of new functionalities, architectural refactoring, and applicable design patterns.

Regarding implementation, the RAG system has been developed using the *LangChain* [1] framework and the *Ollama* [2] inference engine for secure, local LLM execution and the Google Gemini API[3] for high-performance external inference.

The proposal has been validated through a two-phase evaluation. First, the context retrieval performance is evaluated quantitatively with RAGAs [9] to ensure the relevance of the knowledge extracted, achieving a robust F1 score of 0.822. Second, the generation quality is assessed through a comparative study against standard LLMs without context, using a Likert-scale rubric scored by an LLM-as-a-judge approach (combining RAGAs with G-Eval-style [10] prompting) contrasted with expert consensus. Results show that AQUILA is robust and context-aware enough to consistently improve response quality over isolated LLMs.

Based on the validated performance, the primary implication of this proposal is that AQUILA contributes to significantly facilitating the adoption and integration of quantum software in current industrial contexts, offering a technical rigor that general-purpose models cannot achieve independently.

The remainder of this paper is structured as follows. Section 2 reviews the state-of-the-art in quantum software engineering and the use of LLMs. Section 3 details the architecture and foundations of AQUILA. Section 4 describes the technical implementation of AQUILA. Section 5 evaluates AQUILA through a two-phase approach focusing on retrieval and generation quality. Finally, Section 6 summarizes the conclusions and outlines future lines of work.

2 Background and Related Work

The following section reviews the existing literature regarding hybrid software architectures, the use of LLMs in software engineering, and RAG systems.

2.1 Hybrid Software Architectures

QC utilizes counter-intuitive principles such as quantum superposition and entanglement, employing the qubit as its fundamental unit, which can be represented through various physical systems [11]. A qubit does not handle deterministic states of 0 and 1; instead, it considers states in superposition according to probability functions. Thanks to these properties, QC can solve problems that would be intractable for classical computers within a reasonable timeframe, a milestone known as quantum supremacy [12].

However, in practice, maintaining quantum states is extremely difficult. The current state of QC technology relies on noisy, intermediate-scale computers, referred to as NISQ (Noisy Intermediate-Scale Quantum) [1]. In this era, Quantum Processing Units (QPUs) feature a limited number of qubits and high error rates in quantum gates, which results in the partially coherent execution of large quantum circuits. Due to these limitations, researchers have pivoted toward hybrid quantum-classical software

[1] https://docs.langchain.com/oss/python/langchain/overview
[2] https://ollama.com/
[3] https://ai.google.dev/

architectures. For instance, algorithms such as the Quantum Approximate Optimization Algorithm (QAOA) [13] play a significant role. These methods are a cornerstone of the hybrid paradigm, as they split the workflow: QPUs execute the quantum circuits while the classical optimizer updates parameters iteratively.

This paradigm shift moves the problem from physics toward Quantum Software Engineering (QSE) [14, 15]. QSE seeks to apply traditional software engineering principles to the development of complex hybrid systems [16, 17].

One of the major challenges in QSE is the high-level architectural design of hybrid software systems [18], ensuring they can be built efficiently, systematically, and effectively. To address recurring problems in the design of these architectures, the use of design patterns has been proposed. While low-level patterns exist [19, 20], there is a lack of high-level architectural patterns to guide the global design of the system [18].

2.2 LLMs in Software Engineering

Large Language Models (LLMs) represent an evolution in natural language processing and generative artificial intelligence, featuring architectures based on deep neural networks. Trained on massive amounts of data from software repositories and technical documentation, LLMs have not only mastered the syntax of various programming languages but also the logical and semantic relationships inherent in software design.

The emergence of LLMs has revolutionized software engineering (SE) [21, 22]. As documented in several studies [23, 24], the use of LLMs has extended beyond code generation to encompass tasks such as requirements engineering, refactoring, and testing. This surge has consolidated the use of *promptware* in SE to achieve task-specific results [25]. Various projects have materialized the capabilities of LLMs in SE. In terms of implementation, tools such as *AlphaCode* [26] have demonstrated the ability to solve complex programming problems. Regarding maintenance and quality, proposals like *ChatRepair* [27] have automated the repair of logical errors.

In contrast to conventional SE, the use of LLMs in QSE is in a phase of increasing specialization, with recent roadmaps exploring their potential across modeling, generation, testing, and repair [28, 29]. Proposals have emerged that attempt to mitigate domain complexity through fine-tuned models. In the implementation phase, *Qiskit Code Assistant* [3] has shown that, with fine-tuning, models can be effectively employed for quantum code generation and completion. Moving beyond completion assistants, recent initiatives explore autonomous agents capable of translating natural-language prompts directly into executable quantum code, lowering the barrier to quantum programming [30]. LLM-based code analysis has also been explored in comparison to static analyzers such as *LintQ* [31]. In the realm of maintenance, frameworks for the automatic repair of quantum algorithms have been proposed [6].

However, despite these advancements, applying LLMs to specific domains like QC remains a significant challenge. Recent evaluations by O'Brien et al. [7] suggest that true emergent capabilities in software engineering tasks are not guaranteed simply by increasing model scale. This limitation is exacerbated by the complexity of quantum and hybrid systems, where there is often a lack of systematic methods for selecting architectural patterns [32], and up to 82% of quantum software bugs are domain-specific, requiring particular abstractions to be detected [4]. Furthermore, this lack

of domain-specific knowledge can lead to technical hallucinations, where models provide plausible but incorrect answers [8], deviating from both the provided input and physical constraints.

Consequently, the effectiveness of LLMs in complex tasks depends on the quality of the model's knowledge, provided through its context, and the precision of the prompt. This aligns with the vision of future software development, where AI does not replace the engineer; instead, software engineers remain indispensable for understanding, reviewing, and validating AI-generated artifacts [33]. AQUILA facilitates this role by providing the necessary foundation through RAG, thereby enhancing the effectiveness and efficiency of its suggestions.

2.3 Retrieval-Augmented Generation (RAG)

Efforts to improve quantum code generation have recently focused on domain-specific fine-tuning. The *Qiskit Code Assistant* [3] demonstrates that augmenting LLM pre-training with quantum-related documentation outperforms general-purpose LLMs. However, in the quantum domain, this approach necessitates continuous model re-training as datasets are updated to maintain relevance [3].

Recent evidence suggests that providing explicit context can mitigate these limitations and previous failures [5]. Unlike traditional generative models, which rely solely on the knowledge acquired during their training phase, a RAG architecture incorporates external knowledge stored in a database. This information is retrieved based on the user input to provide the additional context required for the response (retrieval phase).

From an architectural perspective, a RAG system is divided into four main components:

- **Knowledge Base Ingestion**: Documents are partitioned into fragments (chunks) and transformed into embeddings (vectors) to capture their semantic meaning. These vectors are then indexed in a vector database.
- **Retrieval**: Upon receiving a query, the system performs a semantic search within the vector database to identify the most relevant chunks based on the input through vector similarity.
- **Augmentation**: The retrieved context is appended to the prompt, creating an augmented query. This ensures the model accounts for the specific context and constraints required for the task.
- **Generation**: The LLM processes the augmented prompt to generate a response, forcing the model to prioritize the evidence found within the retrieved context. This significantly reduces the risk of hallucinations [8].

In software engineering, RAG systems have been adopted for addressing complex and context-dependent tasks. Studies demonstrate their utility in tasks such as repository-level code completion [34], automated software vulnerability repair [35] and test generation [36].

However, within QSE, the adoption of RAG is still in its early stages. EExisting studies have focused on localized code maintenance tasks, such as automating Qiskit

code migration between versions [37] or automated quantum program repair [38]. The systematic use of RAG architectures to guide high-level hybrid software architecture design, structural flaw detection, and design pattern integration remains largely unaddressed.

3 AQUILA

Addressing the challenges and limitations in quantum software engineering regarding the abstract design of hybrid software architectures, the primary contribution of this paper is AQUILA: a RAG system for designing hybrid quantum-classical software architectures while ensuring their quality. The design of AQUILA is based on a standard RAG workflow (see Fig. 1). The process begins with a user query, which is processed by the RAG system to generate an augmented prompt and obtain a response from the LLM.

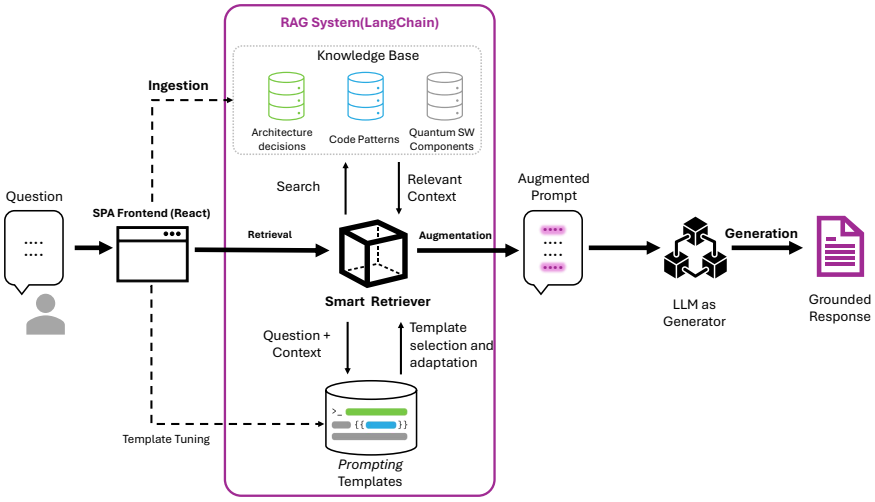


Fig. 1: AQUILA Architectural Workflow

AQUILA's RAG system incorporates the four components previously depicted (knowledge base ingestion, retrieval, augmentation and generation), which are described in detail in Sections 3.1–3.4.

By leveraging this RAG system, AQUILA ensures that the LLM is not merely predicting code based on statistical patterns but is grounded in verified documentation [5]. Through this approach, AQUILA maintains high technical precision and ensures a permanently updated knowledge base for hybrid software, thereby eliminating the need to retrain the base model.

3.1 Knowledge Base Ingestion

The reliability of AQUILA's responses depends on its knowledge base, which integrates unstructured and semi-structured data from various sources. At this layer, the system ingests information from three primary sources: technical documents in PDF format (e.g., design patterns [20] [39], architectural papers [32], manuals, etc.), web resources, and source code files in different languages, specifically *Qiskit* and Python. These code files can be loaded from the local file system or directly from a URL, such as a *GitHub* repository. To transform these diverse sources into a suitable knowledge base, the system employs a processing phase consisting of three sequential stages:

1. **Data Loading:** Using specialized loaders for different context types, the system extracts plain text and metadata. Data loading is implemented through *LangChain* loaders, specifically *PyPDFLoader* and *WebBaseLoader*.
2. **Recursive Chunking:** The plain text is divided into fragments (*chunks*) of uniform size. A default size of 800 *tokens* was selected to achieve an optimal balance between granularity and context. The system allows for flexible configuration between 500 and 1,000 tokens, which can be adjusted in the settings. To ensure coherence and prevent context loss, the system employs a chunk *overlap* strategy, set at 10% of the chunk size, to guarantee thematic continuity between adjacent fragments. The overlap size is also adjustable as a configuration parameter.
3. **Embedding Generation and Storage:** Each chunk is transformed into a high-dimensional vector using an embedding model—in this case, *nomic-embed-text*. The *embeddings* are indexed in a vector database implemented via *ChromaDB*[4], which enables similarity searches using distance metrics.

For source code files, to bridge the technical and structural gap between natural language queries and programming syntax, AQUILA allows indexing a user-provided textual summary or description instead of the raw code content. In this scenario, the semantic search operates over the conceptual description, while retaining the underlying code payload for context injection during augmentation.

For the experiments reported in this study, AQUILA was instantiated with a fixed, domain-specific corpus including the pattern and architectural sources cited above, such as the Pattern Atlas [20], the pattern language for quantum algorithms [39], and the architectural decision material provided by Aktar et al. [32], together with the remaining technical sources listed in the accompanying corpus manifest. The exact composition of this corpus, including source provenance, versions, and access links, is publicly available in the replication package in GitHub [40]. The same corpus was maintained throughout the retrieval and generation evaluations to ensure consistency. The reported results (see section 5) should therefore be interpreted as conditional on this particular knowledge base. The scope, quality, freshness, and internal consistency of the ingested context can affect both the fragments retrieved and the responses generated by the RAG pipeline. The systematic selection, curation, and empirical optimization of the knowledge corpus are outside the scope of this work, which focuses

[4] https://reference.langchain.com/python/langchain-chroma

on AQUILA's conceptual design, the technical feasibility of constructing the proposed system, and a preliminary validation of its retrieval and generation capabilities.

3.2 Retrieval

When the system receives a query, it is converted into a high-dimensional vector using the same *embedding* model employed for knowledge storage. Using this vector, the retriever performs a similarity search to retrieve the k most relevant context chunks from the vector database. This phase, implemented using *LangChain Retrievers*, ensures that the retrieved information forces the language model to ground its response in the provided context.

3.3 Augmentation

Once the relevant fragments are retrieved, they are processed for integration into the specific prompt template selected by the user, ensuring the context is appropriately related to the task. The outcome of this process is an **Augmented Prompt**. As a design decision, a prompt structure has been chosen that concatenates the following elements: an initial behavior instruction defining the LLM's role and response format, the user query, and finally, the retrieved knowledge fragments.

3.3.1 Prompting Templates

The augmentation phase relies on predefined (yet extensible and modifiable) prompting templates which include dynamic fields to be populated and the desired response format. These templates are categorized into two fundamental types:

- **Textual Prompts.** When the user selects a prompt template focused on design, decision-making, or any textual response, AQUILA utilizes a model specialized in text generation.
  - Objective: To provide technical justifications, suggestions, or selection alternatives.
  - Output: Natural language responses enriched with technical references.
- **Code Prompts.** Conversely, when the user selects a template focused on implementation or error mitigation, the system utilizes a model specialized in code generation (e.g., Python/Qiskit code).
  - Objective: To analyze or automate the generation of correct code.
  - Output: Validated code blocks supported by references retrieved from the knowledge base.

These templates are instantiated from a set of high-level conceptual questions designed to guide the LLM to achieve valuable tasks throughout the analysis and design phases of hybrid software architecture development (see Table 1). The underlying LLMs responsible for executing both prompt categories can be dynamically selected by the user from the pool of available models supported by the system.

To transition from the abstract, high-level tasks listed in Table 1 to robust prompting templates, a meta-prompting and optimization process was executed. Rather than

Table 1: Prompt templates for hybrid software architecture tasks

Phase	ID	Task/Prompt Template	Expected Outcome
Analysis	**T1**	How should functionality F be supported in architecture A_C?	Selection of classical/quantum HW with architectural rationale.
	T2	What is the desirable hybrid architecture for F_C given A_C?	Proposal of hybrid architecture A_H.
Design	**T3**	Explain the given hybrid architecture A_H.	Summary linking technical decisions and patterns.
	T4	Does A_H have any issues?	Detection of *architecture smells*.
	T5	How can A_H be refactored to eliminate these smells?	Improved architecture A'_H and refactoring guide.
	T6	Which patterns address quality attribute C or constraint R in A_H?	List of applicable patterns and implementation guide.

asking these conceptual questions to the LLM, this catalog was refactored using a meta-generator model to optimize the instructions.

This process is grounded in state-of-the-art prompt engineering and meta-prompting methodologies. While traditional meta-prompting focuses on dynamic, runtime orchestration where a single LLM acts as a conductor to manage virtual experts [41], AQUILA adapts this philosophy to the design-time template generation phase. This strategy avoids runtime latency overhead and improves consistency in responses related to similar tasks.

Specifically, this meta-prompting strategy structures the final templates around the following fundamental pillars:

1. **Contextual Isolation:** Since AQUILA operates on a RAG architecture [42], the optimized templates introduce strict delimiter boundaries. This prevents the model from conflating retrieved information from its pretrained knowledge, mitigating hallucinations.
2. **Chain of Thought:** To prevent the models from proposing immediate and unverified architectural designs, the prompts force the model to make a step-by-step reasoning [43].
3. **Instruction Optimization:** Employs the LLM as an optimizer to systematically refine prompt phrasing, discovering the most semantically effective instructions to maximize reasoning accuracy on architectural tasks [44].

The practical execution of these fundamentals is realized through a Meta-Prompt to restructure each abstract query into a rigorous template (see Fig. 2).

3.4 Generation

The final phase of the AQUILA workflow is the generation of a grounded response. Crucially, this phase is dictated by initial instructions defined in the system configuration, which establish the baseline rules and grounding requirements for the generation.

```
Act as an expert Prompt Engineer. Your task is to transform a raw, high-level
    software engineering question into a highly structured, production-ready
    prompting template for an LLM, optimized for classical-quantum hybrid
    software architecture tasks.

Input Parameters:
- conceptual_question: The hybrid software engineering task to be achieved.
- expected_outcome:

To generate the optimized template, you must strictly follow these rules:

1. CLEAN STRING-REPLACEMENT PLACEHOLDERS: Every variable must be represented
    ONLY by its placeholder inside single curly braces (e.g., {
    target_functionality}).
   - DO NOT include any trailing instructions, explanations, brackets, or
    examples next to the placeholder.
   - For example:
     * INCORRECT: "Functionality ({target_functionality}): [Insert
    functionality F here]"
     * INCORRECT: "Target: {target_functionality} (e.g., VQE)"
     * CORRECT: "Analyze the following functionality: {target_functionality}"
   The placeholder must stand completely alone as the sole target for
    automated string replacement.

2. STEP-BY-STEP REASONING (CoT): Enforce a strict, logical, step-by-step
    sequence of analysis within the template instructions. Force the model to
    evaluate technical trade-offs (such as classical-quantum communication
    latency, computational complexity, and design pattern applicability)
    before formulating its decision.
-Each step must be a single, direct, and action-oriented sentence focused
    strictly on the core technical evaluation
-Keep the list of steps short and focused only on critical architectural
    decisions.

3. OUTCOME-DRIVEN ALIGNMENT: The generated template must be backward-
    engineered to guarantee the delivery of the exact 'expected_outcome'.
   - The final step of the reasoning sequence must directly synthesize the
    analysis into the required outcome.

Generate the optimized template based on these rules.
```

Fig. 2: Prompt for optimized template generation.

A key feature of this tool is its adaptive capability, where the output format is determined by the specific prompt template selected by the user, providing the appropriate level of abstraction for each task. Due to this capacity, AQUILA supports the software engineering process throughout its entire lifecycle, from analysis and design to implementation, generating two types of responses:

- **Analysis and Architectural Design:** High-level textual rationale, producing textual artifacts such as recommendations, explanations and justifications grounded in the ingested literature.
- **Code synthesis:** Structured, modular code artifacts accompanied by technical annotations and justifications.

This model ecosystem balances local efficiency and low-latency execution with cloud-level reasoning. By incorporating advanced external models such as Google's Gemini alongside local engines, AQUILA significantly extends its context window and token capacity for complex software engineering tasks. Furthermore, the architecture is designed to be model-agnostic, ensuring seamless integration of both local and cloud-based LLMs as model capabilities continue to evolve.

To optimize the performance across diverse tasks, and as previously introduced in Section 3.3, the system allows assigning dedicated LLMs for textual reasoning and code generation independently. Selecting the most suitable model for each task enhances overall generation quality, as models can be chosen based on their specialized strengths, such as architectural rationale or syntax-precise code synthesis.

4 Supporting Tool

The RAG system is implemented as a conversational agent developed using a client-server architecture, featuring a centralized logic layer via a *Flask*-based API and a frontend implemented as a *Single Page Application* (SPA) using *ReactJS* and *Vite* as the build tool. The frontend provides a dedicated workspace for each backend module: conversational menu, configuration, prompt templates, and context storage. Specifically, the implementation leverages the *LangChain* ecosystem, an advanced orchestration framework that enables the modular integration of the necessary RAG components and provides a suite of compatible utilities. The use of *LangChain* is fundamental, as it facilitates seamless data flow and connectivity between the vector database and the language models.

Regarding the inference engine, the system supports a flexible execution architecture. For maximum reasoning performance and high-capacity processing, Google's Gemini is integrated as an external API option. Alternatively, to strictly guarantee data privacy when handling sensitive contextual information, the system leverages *Ollama* for fully local execution. In this privacy-focused setup, open models such as Qwen 2.5 (7b) [5] and Llama 3 (8b) [6] are provided as local LLM alternatives, operating alongside *nomic-embed-text*[7] for vector embeddings generation.

The complete implementation of the supporting tool is publicly available in a GitHub repository [45].

[5] https://ollama.com/library/qwen2.5
[6] https://ollama.com/library/llama3
[7] https://ollama.com/library/nomic-embed-text

4.1 Configuration Parameters

The performance and reliability of AQUILA are closely tied to its internal configuration. The RAG workflow's behavior can be optimized for specific tasks by adjusting hyperparameters that manage the balance between technical precision and generative creativity (see Table 2).

Table 2: AQUILA Configuration Parameters

Parameter	Default	Description
Text Model	Gemini 3.5 Flash	LLM for architectural reasoning and textual synthesis
Code Model	Gemini 3.5 Flash	LLM for code synthesis and generation
Embedding Model	nomic-embed-text	Embedding model for document and query encoding
Temperature	0.1	Controls output creativity. Low value ensures deterministic outputs
Top P	0.9	Probability threshold for candidate token filtering
Top K	40	Restricts token selection to the K most probable candidates
Chunk Size	800 tokens	Maximum length for text segmentation
Overlap	10%	Shared context between contiguous chunks to preserve continuity
System Message	Text string	Agent instructions establishing rules and grounding constraints
Text k	5	Number of retrieved text chunks per query
Code k	3	Number of retrieved code chunks per query

To provide a controlled experimental environment, AQUILA includes a profile management module. Users can adjust and save specific configurations. This profile persistence enables experiment replication and the comparison of response precision across different settings.

4.2 Prompt Templates and Instantiation

As established in the proposed approach, the prompt augmentation relies on the optimized prompt templates, which are the outcome of the meta-prompting process described in Section 3.3. These templates are stored as static assets and exposed directly to the frontend. Users can select the desirable template and populate the dynamic fields to generate the final prompt. To provide a concrete realization of this mechanism, Fig. 3 illustrates an example of one of these optimized templates compared to its original abstract question, including its dynamic fields.

4.3 Context Management

To provide a granular control over the retrieval source, AQUILA incorporates a context management module within its frontend interface. This module enables users to inspect stored context and ingest multi-source documentation, including PDF documents,

```
[CONCEPTUAL QUESTION]

How should functionality F be supported in architecture AC?

======================================================================

[GENERATED PROMPT TEMPLATE]

-Objective
Determine how the following functionality should be supported within the
    following architecture while selecting the appropriate classical and/or
    quantum hardware and providing a justified architectural rationale.

-Inputs
Functionality: {target_functionality}
Architecture: {target_architecture}

-Instructions
Analyze the provided functionality and architecture using the following
    reasoning process.

1. Identify the computational objectives, functional requirements, and
    architectural constraints of {target_functionality} within {
    target_architecture}.
2. Evaluate the computational characteristics of the functionality, including
    algorithmic complexity, parallelism, data dependencies, and execution
    frequency.
3. Assess whether the functionality is better suited for classical computation
    , quantum computation, or a hybrid execution model based on computational
    advantage and hardware capabilities.
4. Evaluate the impact of classical-quantum communication latency, data
    movement overhead, orchestration complexity, and resource availability on
    the architectural decision.
5. Determine the architectural integration approach, including component
    allocation, interaction boundaries, and applicable classical-quantum
    design patterns.
6. Compare the feasible hardware deployment alternatives by evaluating
    performance, scalability, implementation complexity, and architectural
    trade-offs.
7. Select the most appropriate classical and/or quantum hardware configuration
     and justify the decision with a concise architectural rationale.
```

Fig. 3: Comparison between the conceptual question and the optimized, structured prompt template.

web resources via URL, code files and remote code files from GitHub repositories. Specifically, during the ingestion of code files, the interface allows users to supply an

optional high-level description to guide vector indexing, ensuring that complex scripts are indexed by their architectural purpose rather than raw syntax.

A core capability of this module is the dynamic context toggling, which allows users to enable or disable specific documents without deleting them from the vector database. This feature facilitates execution scope adjustments and rapid context switching, enables users to evaluate responses under varying context constraints or conduct comparative experiments without database re-indexing.

As established in Section 3.1, AQUILA was provisioned with a baseline domain-specific corpus. The composition of this corpus is available in GitHub [40].

4.4 Conversational Workspace

The conversational workspace serves as the primary interaction environment, organized into a two-panel layout (see Figure 4). The left panel is dedicated to query preparation and setup, allowing users to select and instantiate prompt templates, attach additional files and select the active execution configuration. The right panel hosts the interactive chat interface, where users submit queries and inspect the generated responses.

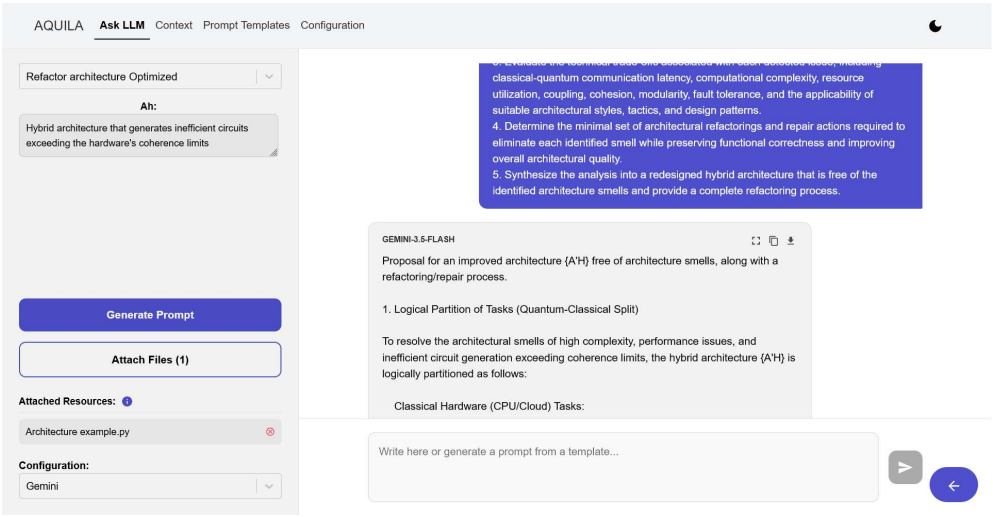


Fig. 4: AQUILA conversational workspace

5 Validation and Results

This section details the experimental validation of AQUILA, focusing on its ability to provide high-quality architectural support for hybrid systems. The evaluation focuses on two phases: the performance assessment of the retrieval engine and the final output generation. First, a quantitative approach based on the *RAGAs* framework [9] is used

to validate the retrieval process (see Section 5.1). For this purpose, ten different queries for quantum software architectures have been developed to ensure the most relevant context chunks are retrieved. Second, the generation stage is evaluated through the outputs for the ten queries by means of expert assessment. This evaluates correctness, completeness, and feasibility, among other features of the AQUILA responses (see Section 5.2).

5.1 Retrieval Evaluation

To evaluate the effectiveness of the retrieval within AQUILA, a multi-case study was conducted focusing on the system's ability to recover relevant context. This evaluation was implemented through a Python-based testing suite integrated with the *RAGAs* framework [9].

5.1.1 Experimental Setup and Dataset

The experimental setup follows the specific configuration parameters detailed in Table 2. Within this environment, the retrieval performance was tested using a set of 10 representative queries covering key concepts of the knowledge base. For each query, a "ground truth" was established by identifying a set of expected context chunk IDs (ranging from 4 to 5 IDs per query) containing the necessary context to make a response to the query. The specific queries developed for this evaluation are listed as follows:

Q1. What are the primary roles and responsibilities of the classical computing layer in a hybrid quantum-classical system?
Q2. Given a hybrid architecture that generates inefficient circuits exceeding the hardware's coherence limits, how can it be redesigned/refactored to eliminate potential problems (architecture smells) by integrating structural approaches and optimization tools?
Q3. Why might traditional modular design fail in quantum software, and how does excessive abstraction affect the physics of the system?
Q4. What are the primary challenges and recommended solutions for deploying hybrid quantum-classical applications?
Q5. To integrate a complex quantum feature into a classical architecture, can Meta-Solving be used to decompose the problem into subproblems and decide whether to solve them classically or quantumly?
Q6. Demonstrate how to integrate error mitigation strategies into the algorithms without altering their underlying core logic.
Q7. Given the severe noise constraints and limited qubit resources of current NISQ hardware, justify the selection of architectural patterns and algorithmic design strategies to ensure the reliability of the quantum module.
Q8. Given an iterative quantum-classical functionality, how should this iterative execution be supported?
Q9. Given a classical application, what architectural patterns should be used to decouple quantum functionalities and expose them as scalable?

Q10. Given a distributed classical software architecture, what is the desirable hybrid execution architecture for quantum functionalities that require deploying Readout Error Mitigation?

During the execution, the system retrieved a set of candidate document IDs for each query. The evaluation script performed a comparative analysis between the retrieved IDs and the expected IDs, calculating two ID-based metrics: *Context Precision* and *Context Recall*.

The complete set of queries and their corresponding ground truth context chunks are publicly available in GitHub [40] to ensure the reproducibility of the evaluation.

5.1.2 Metrics Definition

To provide an objective assessment of the retrieval stage, three primary metrics have been used: precision, recall, and F1 score. In the *RAGAs* framework, precision and recall correspond to *IDBasedContextPrecision* and *IDBasedContextRecall*, respectively, while the F1 score is computed as the harmonic mean of both metrics. These metrics are computed based on the correspondence between the retrieved document IDs and the ground truth IDs.

- **Precision:** This metric evaluates the system's ability to prioritize relevant information fragments over noise. A high score indicates that the model is provided with useful context immediately, optimizing the generation phase.
- **Recall:** This metric evaluates the amount of relevant information retrieved from the total reference fragments. A high score means a higher percentage of the ground truth documents were recovered and not excluded from the final context.
- **F1 score:** This metric provides a balanced assessment of retrieval performance by combining precision and recall through their harmonic mean. A high score indicates that the system achieves both a high proportion of relevant retrieved fragments and a broad coverage of the ground truth documents.

5.1.3 Analysis of Results

The results obtained from the retrieval evaluation are summarized in Table 3. The system was evaluated across ten queries, yielding an Average Context Precision of 0.800 and an Average Context Recall of 0.850. These metrics provide a quantitative foundation for assessing the effectiveness of the retrieval pipeline in a technical domain.

AQUILA achieves a relevant overall performance with an average F1 score of 0.82, indicating a high balance between retrieval precision and recall. This result confirms the system's capacity to precisely identify required context, particularly when queries involve well-defined technical concepts.

It should be noted that perfect scores are obtained for queries such as Q5 and Q7. This is because those queries use more distinctive terms such as "NISQ" or "Meta-Solving". This performance demonstrates the system's capacity to precisely identify the set of required context when the query involves hybrid terminology.

Table 3: Retrieval Performance Metrics (ID-Based)

Query ID	Precision	Recall	F1-score
Q1	0.800	1.000	0.889
Q2	0.600	0.750	0.667
Q3	0.800	0.800	0.800
Q4	0.800	0.800	0.800
Q5	1.000	1.000	1.000
Q6	0.600	0.750	0.667
Q7	1.000	1.000	1.000
Q8	0.800	0.800	0.800
Q9	0.800	0.800	0.800
Q10	0.800	0.800	0.800
Average	**0.800**	**0.850**	**0.822**

In contrast, a closer inspection of the results reveals a minor performance fluctuation in Q2 and Q6, where precision dropped to 0.60 and recall to 0.75. These results are primarily related to the high semantic similarity between various fragments in the knowledge base. While Q7 targets specific constraints, Q2 and Q6 focus on broader tasks like "redesign" or "error mitigation strategies". In these situations, the retriever identified fragments that, while technically relevant to the topic, were not part of the manually selected ground-truth IDs. However, even in these cases, the system prioritized a majority of the relevant context within the top-ranked results.

The average recall of 0.85 is particularly significant for the proposed system. In the context of QC and QSE, missing critical concepts and architectural constraints can lead to design flaws or execution errors. An 85% recovery rate indicates that the majority of the reference knowledge is present in the retrieved context, which may help reduce the risk of architectural hallucinations.

Furthermore, the precision of 0.80 confirms that the retriever effectively manages the limited context windows of the local models by placing the most pertinent fragments at the head of the retrieved context. This result suggests that the retrieval mechanism can provide relevant information to support the generation process.

5.2 Generation Evaluation

While the retrieval was validated quantitatively, the quality of the generated responses was assessed through an automated LLM-as-a-judge validation, contrasted with a *human in the lead* approach [46]. This hybrid evaluation verifies both pipeline grounding and high-level architectural decision-making and ensures that the LLM effectively synthesizes the retrieved context into technical responses.

5.2.1 Evaluation Setup

To quantify the impact of the RAG pipeline on the quality of the results, a comparative study was performed. Each of the queries was submitted to two distinct execution environments:

- **AQUILA (RAG System):** The full RAG-based pipeline utilizing the configuration parameters detailed in Table 2. This setup leverages the knowledge base to provide augmented context to the LLM.
- **LLM without context:** The same Large Language Models operating without the knowledge base or configuration.

The outputs from both environments were then subjected to the same scoring criteria to determine the performance gain provided by the AQUILA as a RAG system against the same LLM used by AQUILA, but without context (Gemini 3.5 Flash). Specifically, this scoring was conducted through an LLM-as-a-judge validation (combining RAGAs metrics [9] for context-specific dimensions with G-Eval-style prompting [10] for response quality criteria) and was contrasted with an independent human expert evaluation.

5.2.2 Evaluation Metrics

To ensure a structured and objective analysis, a set of five evaluation metrics was established. For each query, an expected response (ground truth) was developed. These reference answers served as the standard containing the necessary technical information for a high-quality solution.

The generated output from both approaches, with and without context, were then evaluated by comparing those against the expected responses. The considered metrics are:

- **Technical Correctness:** Measures the accuracy of the generated solution regarding quantum principles and software engineering.
- **Classical/Quantum Partitioning:** Evaluates the logical distribution of tasks between paradigms. This metric assesses if the system correctly identifies which components should remain classical and which should be delegated to the quantum processor.
- **Feasibility:** Evaluates the practical applicability of the response. It considers whether the proposed solution can be implemented on current NISQ hardware and existing software frameworks.
- **Completeness:** Evaluates the extent to which the response addresses all subquestions and constraints present in the query.
- **Context Alignment:** Measures the extent to which the generated response is grounded in the retrieved context, without introducing unsupported or hallucinated information. Unlike correctness, this metric focuses on faithfulness to the retrieved evidence [47].

In the human evaluation, each metric was assessed using a 5-point Likert scale, where 1 indicates poor performance and 5 indicates the best possible performance. For automated evaluation, this 1–5 scale was retained for the first three metrics, whereas *Completeness* and *Context Alignment* were operationalized on a continuous $[0, 1]$ interval. To enable direct comparison between evaluations for *Completeness* and *Context Alignment*, human Likert ratings were linearly normalized via $(S - 1)/4$ to this $[0, 1]$ scale.

5.2.3 Assessment Process

The performance of AQUILA was evaluated across the queries employed in Section 5.1 using the previously defined metrics. For this phase, Gemini 3.5 Flash was employed as the core model to produce the responses. These queries were selected because they are representative of different retrieval and generation conditions observed during the evaluation process, allowing for a more informative comparison than a random or homogeneous set of questions. All evaluated responses correspond to a single execution per query and configuration under a near-deterministic decoding ($T = 0.1$), prioritizing in-depth qualitative evaluation of structural design patterns over statistical sampling of stochastic variations. These queries and their associated evaluation data are available on GitHub [40].

First, an LLM-as-a-judge evaluation was applied to assess the outputs across the established metrics. Specifically, *Context Alignment* and *Completeness* were operationalized using RAGAs [9] *faithfulness* and *answer relevancy* metrics, respectively. These metrics quantitatively measure whether the claims in the response are grounded in the retrieved context and evaluate query alignment through semantic embedding similarity. The remaining three metrics—*Classical/Quantum Partitioning*, *Technical Correctness*, and *Feasibility*—were evaluated following an approach inspired by the G-Eval framework [10], utilizing structured Chain-of-Thought scoring prompts with explicit rubrics. Llama 3.1:8b was employed for the RAGAs evaluation, whereas GPT-6 served as the judge for the multi-criteria assessment due to its advanced reasoning capabilities.

Second, to contrast these automated results and account for the potential limitations and sensitivities of automated judges, a human expert evaluation was conducted. Evaluators scored the outputs without prior knowledge of the LLM-as-a-judge ratings. To reduce subjectivity and improve inter-evaluator consistency, the assessment followed a structured consensus-based procedure, conceptually aligned with the Delphi approach: the authors first evaluated the outputs independently and then discussed score discrepancies until reaching a shared judgment.

5.2.4 Result Analysis

The results of both evaluation methods (human evaluation and llm-as-a-judge) are summarized in Figure 5, while Table 4 reports the per-question score differences across all evaluated criteria.

As detailed in Table 4, both evaluation methods exhibit strong alignment and consistent scoring trends, showing that AQUILA generally matches or outperforms the LLM without context, with only isolated exceptions. To avoid over-optimistic claims,

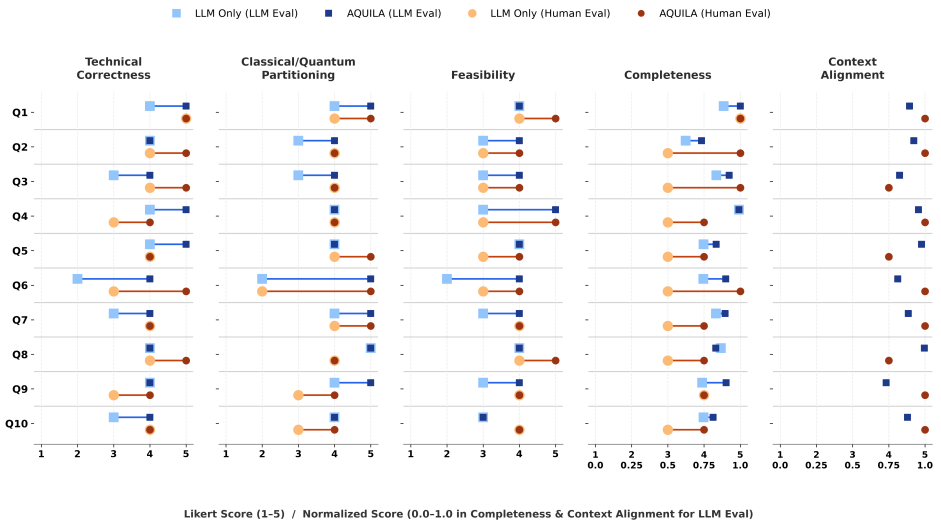


Fig. 5: Comparative evaluation results between LLM without context and AQUILA RAG system across LLM-as-a-judge and human evaluation paradigms

Table 4: Summary of per-question score differences between AQUILA and the LLM-only baseline across evaluation criteria and evaluator types.

Evaluation Metric	Evaluator	#Questions ($n = 10$)			Score Difference (Δ)		
		Higher	Equal	Lower	Mean	Min	Max
Technical Correctness	LLM	7	3	0	+0.80	0	+2
	Human	6	4	0	+0.70	0	+2
C/Q Partitioning	LLM	6	4	0	+0.80	0	+3
	Human	6	4	0	+0.80	0	+3
Feasibility	LLM	6	4	0	+0.80	0	+2
	Human	7	3	0	+0.80	0	+2
Completeness	LLM	8	1	1	+0.08	-0.04	+0.17
	Human	8	2	0	+0.28	0	+0.50

Note: Δ denotes the per-question score difference between AQUILA and the LLM-only baseline ($\Delta = S_{\text{AQUILA}} - S_{\text{LLM-only}}$). Higher, Equal, and Lower indicate the number of questions for which AQUILA achieves a higher, equal, or lower score than the baseline, respectively. Mean, Min, and Max represent the mean, minimum, and maximum score differences across the ten questions. The first three criteria use a 1–5 Likert scale. For Completeness, human evaluation scores (originally on a 1–5 scale) are normalized via $(S - 1)/4$ to match the [0,1] scale used by the LLM evaluator. Context Alignment is excluded as it is not applicable to the LLM-only baseline.

we adopt the LLM-as-a-judge evaluation as our primary and conservative reporting reference. The automated judge yields a lower, less optimistic bound, most notably in *Completeness*, where it reports a more conservative mean of +0.08 compared to the normalized +0.28 assigned by human evaluators.

Under this conservative stance, AQUILA achieves consistent gains across all dimensions. For the 1–5 Likert metrics, AQUILA achieves an average improvement of +0.80 in *Technical Correctness* (with gains of up to +2 across 7 queries), an identical average gain in *Classical/Quantum Partitioning* (reaching peaks of up to +3 points), and +0.80 in *Feasibility*, gaining up to +2 across 6 queries. In terms of *Completeness*, AQUILA surpasses the baseline in 8 out of 10 queries, with a mean difference of +0.08 on a normalized $[0, 1]$ scale. Furthermore, although *Context Alignment* is not applicable to the baseline, AQUILA demonstrates robust context fidelity, achieving an average score of 0.89 in automated evaluation and 4.70/5 in human assessment. Crucially, these collective improvements demonstrate that domain-specific context prevents superficial responses and improves architectural decisions.

In highly theoretical queries, such as Q1 (roles and responsibilities of the classical layer) and Q7 (architectural patterns for reliability), the standalone Gemini 3.5 Flash exhibits a strong baseline performance. In Q1, the automated judge assigned the baseline a solid 4 across these metrics while awarding AQUILA a perfect score. Even in scenarios where the base model is competent, AQUILA provides valuable refinements.

Conversely, more pronounced performance differences are observed in applied queries requiring concrete domain-specific context. The most substantial disparity occurs in Q6 (integration of error mitigation strategies). In this scenario, the LLM without context struggles to establish clear classical-quantum separation, dropping to a score of 2 in several metrics. In contrast, AQUILA achieves high scores in *Technical Correctness*, *Partitioning*, *Completeness*, and *Context Alignment*, while additionally enhancing *Feasibility*, highlighting the utility of specialized context.

A deeper inspection of Q6 clearly illustrates the cause behind the observed performance gap, as summarized in Figure 6: the LLM without context suffered from severe domain conflation. Although it correctly identified the design patterns, it treated quantum error mitigation as classical exception handling around a standard function. Due to this failure to recognize the quantum paradigm, both evaluation methods heavily penalized the baseline, especially in *Technical Correctness*, *Classical/Quantum Partitioning* and *Feasibility*, as classical software retries do not mitigate physical quantum noise. In contrast, AQUILA's response provides a more quantum-specific architectural interpretation of the problem. It explicitly identifies classical and quantum responsibilities, placing mitigation logic on the classical side while delegating circuit execution to the QPU. Furthermore, AQUILA addresses the performance, complexity and scalability trade-offs, representing a more appropriate interpretation of error mitigation, while preserving the core algorithm.

Crucially, across all evaluated queries, regardless of their theoretical or applied nature, a key advantage of integrating AQUILA with large cloud-based models is its ability to consistently anchor generated responses in domain-specific context. While standalone LLMs tend to generate generic recommendations, specialized context ensures that architectural advice precisely aligns with target domain constraints.

Beyond cloud-based models, an additional evaluation was performed using Llama 3 in a local execution environment. While this section highlights the results for Gemini 3.5 Flash, the complete experimental dataset and scoring breakdown for Llama 3 are available in the repository [40]. It should be noted that benchmarking AQUILA

(a) Gemini 3.5 Flash without context

Approach: Pure classical exception handling.
Partitioning: Monolithic classical space (no QPU awareness).
Mechanisms: Retries, backoff, and fallback masking.

```python
# Domain conflation: classical exception trapping
def mitigate_errors(max_retries=3, fallback=0.0):
    def decorator(core_algorithm):
        def wrapper(*args, **kwargs):
            for attempt in range(1, max_retries + 1):
                try:
                    return core_algorithm(*args, **kwargs)
                except Exception as error:  # Retry
                    time.sleep(0.2 * (2 ** attempt))
            return fallback
        return wrapper
    return decorator

# Evaluated core: classical function
@mitigate_errors(max_retries=3)
def compute_moving_average(data: list) -> float:
    return sum(data) / len(data)
```

Issue: Software retries cannot mitigate physical quantum noise.

(b) AQUILA response (RAG Pipeline)

Approach: Quantum Patterns + Decorator.
Partitioning: Classical host (pre/post-process) vs. QPU backend.
Mechanisms: Gate Error Mitigation and Readout calibration.

```python
class ErrorMitigationDecorator:
    """Decouples noise from core quantum logic."""
    def __init__(self, alg, gate_mit, ro_mit):
        self._alg = alg
        self._gate_mit = gate_mit
        self._ro_mit = ro_mit

    def execute(self, circuit, backend):
        # 1. Classical Pre-processing (Folding)
        if self._gate_mit:
            circuit = self._gate_mit.apply(circuit)
        # 2. Physical QPU Execution
        raw = self._alg.execute(circuit, backend)
        # 3. Classical Post-processing (Readout)
        if self._ro_mit:
            return self._ro_mit.calibrate(raw)
        return raw
```

Strength: Principled hybrid split; preserves core algorithm on QPU.

Fig. 6: Synthesized overview of key differences and representative code excerpts for query Q6 (error mitigation decoupling).

against a cloud model using public web-derived data represents a highly conservative evaluation baseline, as such models have already ingested open-source quantum documentation. Consequently, the observed improvements reflect strict architectural alignment and hallucination mitigation rather than mere knowledge retrieval.

Overall, these results provide preliminary evidence that AQUILA's integrated RAG-based configuration can improve response quality and reduce incomplete or technically inappropriate architectural recommendations under the evaluated conditions.

5.3 Threats to Validity

This study is subject to several threats to validity. First, regarding construct validity, the retrieval evaluation relies on manually defined ground-truth chunk IDs. This operationalization enables a reproducible comparison through precision, recall, and F1 score, but it may not fully represent the semantic relevance of the retrieved context. In particular, a retrieved fragment may provide technically valid information even when its identifier does not match one of the manually selected reference chunks, potentially underestimating retrieval performance. Conversely, the manual construction of the ground truth may be influenced by the researchers' interpretation of which fragments are necessary to answer each query. This threat was mitigated by defining the reference chunks using domain knowledge and by complementing the ID-based retrieval metrics

with a generation-oriented assessment. Nevertheless, alternative relevance judgments or semantic evaluation criteria could produce different results.

Second, internal validity may be affected by subjective judgment. Since the authors participated in both the development of AQUILA and the evaluation of its outputs, the assigned scores may be influenced by observer bias or prior knowledge of the evaluated approach. To reduce this risk, two complementary measures were adopted: first, an automated LLM-as-a-judge evaluation (leveraging RAGAs and a G-Eval-based prompt) was conducted to provide reproducible scoring decoupled from the authors' subjectivity; second, to contrast these automated results against domain expertise, a human assessment was subsequently carried out. The human assessment was conducted blindly without prior knowledge of the automated judge's scores, and relied on predefined evaluation criteria and reference responses. The authors initially scored the outputs independently before resolving disagreements through a structured consensus process. Additionally, the intrinsic stochasticity of generative models poses a threat to reproducibility; to mitigate output variance and prioritize technically focused responses, generation was configured with a low temperature ($T = 0.1$), though minor non-deterministic variability may still persist. Nevertheless, because automated judges may exhibit sensitivities to prompt formulations and non-determinism, future evaluations should therefore involve external domain experts, fully blinded comparative setups, and explicit measures of inter-rater agreement.

Third, external validity is limited by the size and scope of the evaluation. The generation study was conducted on ten representative queries. Crucially, the reported results reflect the evaluation of AQUILA not as an isolated retrieval engine, but as an integrated configuration comprising the proposed framework, its task-specific prompt templates, and a concrete, custom-curated knowledge corpus focused on hybrid quantum-classical architectures and quantum software engineering [40]. Moreover, system performance is conditioned by the specific corpus used to instantiate the knowledge base, since the selection, quality, coverage, freshness, and internal consistency of the ingested sources may affect both the retrieved fragments and the generated responses. Consequently, different source collections or curation decisions could lead to different retrieval and generation outcomes. The systematic selection, curation, and optimization of the knowledge corpus are outside the scope of this work, which focuses on AQUILA's conceptual design, the feasibility of constructing the proposed system, and its preliminary validation. As a result, the findings should be interpreted as evidence within this domain, corpus, and experimental setting, rather than as broadly generalizable results. Future work should extend the evaluation to a larger number of queries, evaluators, and heterogeneous and independently curated corpora.

Finally, conclusion validity is constrained by the exploratory nature of the generation evaluation. The assessment is based on a limited number of queries and on scores collected using a five-point Likert scale. Although corroborating the results across two complementary methods (human consensus and automated judges) strengthens credibility, the small sample and ordinal ratings mean that the aggregated scores are intended as descriptive indicators for comparing the two configurations rather than as evidence of statistically significant differences. In addition, the consensus process produces a final shared score but does not quantify the initial level of agreement among

evaluators. Accordingly, the reported results should be interpreted as preliminary evidence of the benefits of the RAG-based configuration. Future studies should employ larger samples, retain individual evaluator scores, report inter-rater reliability, and apply appropriate statistical analyses to assess the magnitude and significance of the observed differences.

6 Conclusions and Future Work

This paper has presented AQUILA, a RAG system for the design and enhancement of hybrid software architectures, based on a technical knowledge base comprising design patterns and specialized literature. The results provide preliminary evidence that integrating vector databases with LLMs provides the necessary technical support for designing and improving complex hybrid software architectures, overcoming the limitations of general-purpose LLMs in high-precision tasks.

The feasibility of this proposal has been assessed through a two-phase validation process. The results, characterized by a retrieval F1 score of 0.822 and favorable evaluations of generation quality, provide preliminary evidence that AQUILA can generate coherent, technically grounded responses and may improve response quality compared with non-augmented models under the evaluated conditions.

Despite these promising, preliminary results, the proposal presents certain limitations. Currently, cases of suboptimal recovery detected in the validation highlight that the system relies on the LLM's internal reasoning to reconcile retrieved information, as it lacks an explicit mechanism to detect or filter contradictions between recovered fragments. Furthermore, when executing local LLMs to preserve privacy, the performance is bounded by the context capacity of local open models, which restricts the amount of retrieved context and the complexity of manageable queries. These limitations draw the work in progress and future research lines as follows:

- **Transition toward Graph-based RAG (*GraphRAG*)**, which would allow the system to capture complex interdependencies and structural relationships within the technical contexts.
- **Model Context Protocol (MCP) integration**, enabling standardized access to external tools and knowledge sources to complement the existing retrieval pipeline and support more flexible context integration.
- **Large-scale empirical validation**, expanding the current evaluation to a significantly larger number of queries and incorporating external assessment to strengthen validity.

Despite these limitations, AQUILA has relevant implications for quantum software engineering. By combining augmented prompts with context retrieved from a specialized and updatable knowledge base, AQUILA supports more informed architectural decision-making and improves the technical grounding and consistency of hybrid software designs. This capability may facilitate the modernization of legacy systems by guiding their evolution toward hybrid quantum-classical architectures. Furthermore,

its model-agnostic design and decoupled knowledge layer allow the framework to incorporate emerging LLMs and evolving quantum computing knowledge without requiring substantial changes to the core workflow, thereby enhancing its adaptability and long-term maintainability. Overall, AQUILA provides a feasible foundation for supporting the industrial adoption of quantum software through the more systematic application of hybrid software engineering practices.

Acknowledgment

This work has been supported by the SMOOTH (PID2022-137944NB-I00) and ATHENA-HYGIEIA (PID2024-155693NB-C42) projects, funded by MICIU/AEI/10.13039/501100011033/PRTR, European Union; by the 2025-GRIN-38310 (SINERGIA) project, under the UCLM Research Plan, co-funded (85%) by the European Regional Development Fund (ERDF).

References

[1] Preskill, J.: Quantum Computing in the NISQ era and beyond. Quantum **2**, 79 (2018) https://doi.org/10.22331/q-2018-08-06-79

[2] Kim, Y., Eddins, A., Anand, S., Takita, M., Brink, B., McManus, N., Janssen, S., Gunluk, O., Nayfeh, H., Gebre, I., *et al.*: Evidence for the utility of quantum computing before fault tolerance. Nature **618**(7965), 500–505 (2023) https://doi.org/10.1038/s41586-023-06096-3

[3] Dupuis, N., Buratti, L., Vishwakarma, S., Forrat, A.V., Kremer, D., Faro, I., Puri, R., Cruz-Benito, J.: Qiskit Code Assistant: Training LLMs for generating Quantum Computing Code. arXiv. arXiv:2405.19495 [quant-ph] (2024). https://doi.org/10.48550/arXiv.2405.19495 . http://arxiv.org/abs/2405.19495

[4] Paltenghi, M., Pradel, M.: Analyzing Quantum Programs with LintQ: A Static Analysis Framework for Qiskit. Proceedings of the ACM on Software Engineering **1**(FSE), 2144–2166 (2024) https://doi.org/10.1145/3660802

[5] d'Aloisio, G., Fortz, S., Hanna, C., Fortunato, D., Bensoussan, A., Mendiluze Usandizaga, E., Sarro, F.: Exploring LLM-Driven Explanations for Quantum Algorithms. In: Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering And Measurement, pp. 475–481. ACM, Barcelona Spain (2024). https://doi.org/10.1145/3674805.3690753 . https://dl.acm.org/doi/10.1145/3674805.3690753

[6] Yoshida, C., Ishimoto, Y., Nourry, O., Kondo, M., Matsushita, M., Kamei, Y., Higo, Y.: Leveraging Mutation Analysis for LLM-based Repair of Quantum Programs (2026). https://arxiv.org/abs/2601.12273

[7] O'Brien, C., Rodriguez-Cardenas, D., Velasco, A., Palacio, D.N., Poshyvanyk, D.: Measuring Emergent Capabilities of LLMs for Software Engineering: How Far Are We? arXiv. arXiv:2411.17927 [cs] (2024). https://doi.org/10.48550/arXiv.2411.17927 . http://arxiv.org/abs/2411.17927

[8] Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y., Chen, D., Dai, W., Chan, H.S., Madotto, A., Fung, P.: Survey of Hallucination in Natural Language Generation. ACM Computing Surveys **55**(12), 1–38 (2023) https://doi.org/10.1145/3571730 . arXiv:2202.03629 [cs]

[9] Es, S., James, J., Espinosa Anke, L., Schockaert, S.: RAGAs: Automated evaluation of retrieval augmented generation. In: Aletras, N., De Clercq, O. (eds.) Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations, pp. 150–158. Association for Computational Linguistics, St. Julians, Malta (2024). https://doi.org/10.18653/v1/2024.eacl-demo.16 . https://aclanthology.org/2024.eacl-demo.16/

[10] Liu, Y., Iter, D., Xu, Y., Wang, S., Xu, R., Zhu, C.: G-eval: NLG evaluation using gpt-4 with better human alignment. In: Bouamor, H., Pino, J., Bali, K. (eds.) Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, pp. 2511–2522. Association for Computational Linguistics, Singapore (2023). https://doi.org/10.18653/v1/2023.emnlp-main.153 . https://aclanthology.org/2023.emnlp-main.153/

[11] Ezratty, O.: Understanding Quantum Technologies, (2025)

[12] Arute, F., Arya, K., Babbush, *et al.*: Quantum supremacy using a programmable superconducting processor. Nature **574**(7779), 505–510 (2019) https://doi.org/10.1038/s41586-019-1666-5

[13] Farhi, E., Goldstone, J., Gutmann, S.: A Quantum Approximate Optimization Algorithm. arXiv. arXiv:1411.4028 [quant-ph] (2014). https://doi.org/10.48550/arXiv.1411.4028 . http://arxiv.org/abs/1411.4028

[14] Piattini, M., Hevia Oliver, J.L., Pérez-Castillo, R., Serrano, M., Peterssen Nodarse, G.: Toward a quantum software engineering. IT Professional **23**, 62–66 (2021) https://doi.org/10.1109/MITP.2020.3019522

[15] Zhao, J.: Quantum Software Engineering: Landscapes and Horizons (2021). https://arxiv.org/abs/2007.07047

[16] Serrano, M., Pérez-Castillo, R., Piattini, M., Peterssen Nodarse, G.: Quantum Software Engineering, p. 302. Springer, ??? (2022)

[17] Murillo, J.M., Garcia-Alonso, J., Moguel, E., Barzen, J., Leymann, F., Ali, S., Yue, T., Arcaini, P., Pérez-Castillo, R., Guzmán, I., Piattini, M., Ruiz-Cortés, A., Brogi, A., Zhao, J., Miranskyy, A., Wimmer, M.: Quantum software engineering: Roadmap and challenges ahead. ACM Transactions on Software Engineering and Methodology **34**(5), 1–48 (2025) https://doi.org/10.1145/3712002

[18] Khan, A.A., Ahmad, A., Waseem, M., Liang, P., Fahmideh, M., Mikkonen, T., Abrahamsson, P.: Software Architecture for Quantum Computing Systems – A Systematic Review (2023). https://arxiv.org/abs/2202.05505

[19] Bühler, F., Barzen, J., Beisel, M., Georg, D., Leymann, F., Wild, K.: Patterns for Quantum Software Development. In: Proceedings of the 15th International Conference on Pervasive Patterns and Applications, pp. 30–39. Xpert Publishing Services (XPS), (2023)

[20] Leymann, F., Barzen, J.: In: Aiello, M., Bouguettaya, A., Tamburri, D.A., Heuvel, W.-J. (eds.) Pattern Atlas, pp. 67–76. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-73203-5_5 . https://doi.org/10.1007/978-3-030-73203-5_5

[21] Bhati, H.: Agentic AI in the Software Development Lifecycle: Architecture, Empirical Evidence, and the Reshaping of Software Engineering. arXiv. arXiv:2604.26275 [cs.SE] (2026). https://doi.org/10.48550/arXiv.2604.26275 . http://arxiv.org/abs/2604.26275 Accessed 2026-07-15

[22] Hassan, A.E., Li, H., Lin, D., Adams, B., Chen, T.-H., Kashiwa, Y., Qiu, D.: Agentic Software Engineering: Foundational Pillars and a Research Roadmap. arXiv. arXiv:2509.06216 [cs.SE] (2026). https://doi.org/10.48550/arXiv.2509.06216 . http://arxiv.org/abs/2509.06216 Accessed 2026-07-15

[23] Zhang, Q., Fang, C., Xie, Y., Zhang, Y., Yang, Y., Sun, W., Yu, S., Chen, Z.: A Survey on Large Language Models for Software Engineering. arXiv. arXiv:2312.15223 [cs] (2024). https://doi.org/10.48550/arXiv.2312.15223 . http://arxiv.org/abs/2312.15223

[24] Liu, J., Wang, K., Chen, Y., Peng, X., Chen, Z., Zhang, L., Lou, Y.: Large Language Model-Based Agents for Software Engineering: A Survey. arXiv. arXiv:2409.02977 [cs] (2025). https://doi.org/10.48550/arXiv.2409.02977 . http://arxiv.org/abs/2409.02977

[25] Chen, Z., Wang, C., Sun, W., Liu, X., Zhang, J.M., Liu, Y.: Promptware Engineering: Software Engineering for Prompt-Enabled Systems. arXiv. arXiv:2503.02400 [cs] (2026). https://doi.org/10.48550/arXiv.2503.02400 . http://arxiv.org/abs/2503.02400

[26] Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, *et al.*: Competition-level code generation with alphacode. Science **378**(6624), 1092–1097 (2022) https://doi.org/10.1126/science.abq1158

[27] Xia, C.S., Zhang, L.: Keep the conversation going: Fixing 162 out of 337 bugs for $0.42 each using chatgpt. In: Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), pp. 524–536 (2023)

[28] Porta, A.D., Lambiase, S., Lucia, A.D., Palomba, F.: Are Large Language Models Ready for Quantum Software Engineering? A Multivocal Literature Review. Research Square. ISSN: 2693-5015 (2026). https://doi.org/10.21203/rs.3.rs-10126259/v1 . https://www.researchsquare.com/article/rs-10126259/v1 Accessed 2026-07-28

[29] Ye, J., Hu, Q., Zhang, F., Xia, S., Guo, X., Wu, X., Lyu, Y., Zhao, J.: Large Language Models for Quantum Software Engineering: Opportunities and Challenges in Modeling, Generation, Testing, and Repair. ACM Transactions on Software Engineering and Methodology, 3828167 (2026) https://doi.org/10.1145/3828167 . Accessed 2026-07-22

[30] Castelvecchi, D.: Ai tool lets researchers 'vibe code' in the quantum realm. Nature

656(8129), 800–801 (2026) https://doi.org/10.1038/d41586-026-02495-4

[31] Shin, S.Y., Pastore, F., Bianculli, D.: Quantum Program Linting with LLMs: Emerging Results from a Comparative Study. arXiv. arXiv:2504.05204 [cs] (2025). https://doi.org/10.48550/arXiv.2504.05204 . http://arxiv.org/abs/2504.05204

[32] Aktar, M.S., Liang, P., Waseem, M., Tahir, A., Shahin, M., Akbar, M.A., Khan, A.A., Ahmad, A., Dieu, M.J.d., Li, R.: Decision Models for Selecting Architecture Patterns and Strategies in Quantum Software Systems. arXiv. arXiv:2507.11671 [cs] (2025). https://doi.org/10.48550/arXiv.2507.11671 . http://arxiv.org/abs/2507.11671

[33] Terragni, V., Vella, A., Roop, P., Blincoe, K.: The Future of AI-Driven Software Engineering. ACM Transactions on Software Engineering and Methodology **34**(5), 1–20 (2025) https://doi.org/10.1145/3715003

[34] Zhang, F., Chen, B., Zhang, Y., Keung, J., Liu, J., Zan, D., Mao, Y., Lou, J.-G., Chen, W.: RepoCoder: Repository-Level Code Completion Through Iterative Retrieval and Generation (2023). https://arxiv.org/abs/2303.12570

[35] Gadey, V., Liu, Z., Dmitrienko, A.: RAVEN: Agentic RAG for Automated Vulnerability Repair (2026). https://arxiv.org/abs/2606.22647

[36] Shin, J., Harzevili, N.S., Aleithan, R., Hemmati, H., Wang, S.: Retrieval-Augmented Test Generation: How Far Are We? (2025). https://arxiv.org/abs/2409.12682

[37] Suarez, J.M., Bibbo, L.M., Bogado, J., Fernandez, A.: Qiskit Code Migration with LLMs (2026). https://arxiv.org/abs/2606.20173

[38] Le, X.-B., Tran-Truong, P.T., Nguyen, D.: When grounding hurts: Retrieval and validation for llm repair of quantum programs. In: Proceedings of the 30th International Conference on Engineering of Complex Computer Systems (ICECCS 2026), Brisbane, Australia (2026)

[39] Leymann, F.: Towards a pattern language for quantum algorithms. In: Quantum Technology and Optimization Problems. Lecture Notes in Computer Science (LNCS), vol. 11413, pp. 218–230. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-14082-3_19

[40] Espada, F.J.: AQUILA-resultados. https://github.com/fjespada/AQUILA-resultados (2026)

[41] Suzgun, M., Kalai, A.T.: Meta-Prompting: Enhancing Language Models with Task-Agnostic Scaffolding (2024). https://arxiv.org/abs/2401.12954

[42] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H.,

Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., Kiela, D.: Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. arXiv. arXiv:2005.11401 [cs] (2021). https://doi.org/10.48550/arXiv.2005.11401 . http://arxiv.org/abs/2005.11401

[43] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., Zhou, D.: Chain-of-Thought Prompting Elicits Reasoning in Large Language Models (2023). https://arxiv.org/abs/2201.11903

[44] Yang, C., Wang, X., Lu, Y., Liu, H., Le, Q.V., Zhou, D., Chen, X.: Large Language Models as Optimizers (2024). https://arxiv.org/abs/2309.03409

[45] Espada, F.J.: AQUILA. https://github.com/FranEspada13/AQUILA (2026)

[46] Packowski, S., Halilovic, I., Schlotfeldt, J., Smith, T.: Optimizing and Evaluating Enterprise Retrieval-Augmented Generation (RAG): A Content Design Perspective. arXiv. arXiv:2410.12812 [cs] (2024). https://doi.org/10.48550/arXiv.2410.12812 . http://arxiv.org/abs/2410.12812

[47] Wallat, J., Heuss, M., Rijke, M., Anand, A.: Correctness is not Faithfulness in RAG Attributions (2024). https://arxiv.org/abs/2412.18004